# rituals Documentation

***Release 0.3.0***

**Jürgen Hermann**

2015-12-15

Contents


**1 Overview** **1**

**2 Important Links** **3**

**3 Documentation Contents** **5**
- 3.1 User's Manual . . . 5
- 3.2 Tasks Reference . . . 8
- 3.3 Configuration Reference . . . 9
- 3.4 Complete API Reference . . . 10
- 3.5 Contribution Guidelines . . . 19
- 3.6 Software License . . . 20

**4 References** **29**
- 4.1 Tools . . . 29

**5 Indices and Tables** **31**

**Python Module Index** **33**

CHAPTER 1

# Overview

"Rituals" is a task library for Invoke that keeps the most common tasks you always need out of your project, and makes them centrally maintained. This leaves your `tasks.py` small and to the point, with only things specific to the project at hand.

For more information, continue with the User's Manual.

CHAPTER 2

# Important Links

- GitHub Project
- Issue Tracker
- PyPI
- Latest Documentation
- Google Group

CHAPTER 3

# Documentation Contents

## 3.1 User's Manual

### 3.1.1 Introduction

"Rituals" is a task library for Invoke that keeps the most common tasks you always need out of your project, and makes them centrally maintained. This leaves your `tasks.py` small and to the point, with only things specific to the project at hand.

The following lists the common task implementations that the `rituals.easy` module offers. See *below* on how to integrate them into your `tasks.py`.

- `help` – Default task, when invoked with no task names.
- `clean` – Perform house-cleaning.
- `build` – Build the project.
- `docs` – Build the documentation.
- `test` – Perform standard unittests.
- `check` – Perform source code checks.
- `release.bump` – Bump a development version.
- `release.dist` – Distribute the project.
- `release.prep` – Prepare for a release.
- ... and *many* more, see `inv -l` for a complete list.

The guiding principle for these tasks is to strictly separate low-level tasks for building and installing (via `setup.py`) from high-level convenience tasks a developer uses (via `invoke`). *Invoke* tasks can use *Setuptools* ones as building blocks, but never the other way 'round – this avoids any bootstrapping headaches during package installations.

Use `inv -h ‹task›` as usual to get details on the options of these tasks. The Tasks Reference explains them in more detail. Look at the modules in rituals.acts if you want to know every nuance of what these tasks do.

**Note:**

The easiest way to get a working project using `rituals` is the py-generic-project cookiecutter archetype, which is tightly integrated with the tasks defined here.

That way you have a working project skeleton within minutes that is fully equipped, with all aspects of building, testing, quality checks, continuous integration, documentation, and releasing covered.

---

### 3.1.2 Adding Rituals to Your Project

First of all, include `rituals` as a dependency in your `dev-requirements.txt` or a similar file, to get a release from PyPI. To refer to the current GitHub `master` branch instead, use a `pip` requirement like this:

```
-e git+https://github.com/jhermann/rituals.git#egg=rituals
```

Then at the start of your `tasks.py`, use the following statement to define *all* tasks that are considered standard:

```
from rituals.easy import *
```

This works by defining the `namespace` identifier containing Ritual's default tasks. Note that it also defines Invoke's `Collection` and `task` identifiers, and some other common helpers assembled in *`rituals.easy`*. Rituals' own tasks.py can serve as an example.

Of course, you may also do more selective imports, or build your own *Invoke* namespaces with the specific tasks you need.

**Warning:** These tasks expect an importable `setup.py` that defines a `project` dict with the setup parameters, see rudiments and py-generic-project for examples. The needed changes are minimal:

```
project = dict(  # this would usually be a setup(...) call
    name='...',
    ...
)
if __name__ == '__main__':
    setup(**project)
```

### 3.1.3 Task Namespaces

#### The Root Namespace

The tasks useful for any (Python) project are organized in a root namespace When you use the `from rituals.easy import *` statement, that also imports this root namespace. By convention of *Invoke*, when the identifier `namespace` is defined, that one is taken instead of constructing one automatically from all defined tasks.

It contains some fundamentals like `clean`, and nested namespaces handling specific topics. Examples of nested namespaces are `test`, `check`, `docs`, and `release`. See Tasks Reference for a complete list.

The root namespace has `help` as the default task, and most nested namespaces also have a default with the most commonly performed action. These default tasks are automatically aliased to the name of the namespace, so for example `docs.sphinx` can also be called as `docs`.

#### Adding Local Task Definitions

Having an explicit root namespace means that within `tasks.py`, you need to register your own tasks using its `add_task` method, if you want them to be available as top-level names:

```
@task
def my_own_task(ctx):
    """Something project-specific."""
    ...

namespace.add_task(my_own_task)
```

Rituals' own tasks.py uses this to add some local tasks.

Another strategy is to add them in bulk, so when you write a new task you cannot forget to make it visible:

```
# Register local tasks in root namespace
from invoke import Task
for _task in globals().values():
    if isinstance(_task, Task) and _task.body.__module__ == __name__:
        namespace.add_task(_task)
```

Add the above snippet to the *end* of your `tasks.py`, and every *local* task definition gets added to the root namespace.

#### Constructing Your Own Namespace

When you want to have more control, you can exclude the `namespace` identifier from the import and instead define your own. This example taken from py-generic-project shows how it's done:

```
from rituals.easy import task, Collection
from rituals.acts.documentation import namespace as _docs

...

namespace = Collection.from_module(sys.modules[__name__], name='')
namespace.add_collection(_docs)
```

Note that the `name=''` makes this a root namespace. If you need to be even more selective, import individual tasks from modules in *`rituals.acts`* and add them to your namespaces.

### 3.1.4 How-Tos

#### Change default project layout

By default, sources are expected in `src/‹packagename›` and tests in `src/tests`.

You can change this by calling one of the following functions, directly after the import from `rituals.invoke_tasks`.

- `config.set_maven_layout()` – Changes locations to `src/main/python/‹packagename›` and `src/test/python`.
- `config.set_flat_layout()` – Changes locations to `‹packagename›` and `tests`.

#### Change default project configuration

**TODO**

## 3.2 Tasks Reference

Please make sure you also read the section on *Task Namespaces*.

### 3.2.1 Fundamental Tasks

#### Getting Help

`help` is the default task in the root namespace, so that just calling `inv` gives a reasonable response. It combines the global help (`inv -h`) and the task listing (`inv -l`).

#### Project Cleanup

The `clean` task gets rid of various groups of generated files, selectable by options. These options are:

```
-a, --all         The same as --backups --bytecode --dist --docs
-b, --backups     Also clean '*~' files etc.
-d, --docs        Also clean the documentation build area
-e, --extra       Any extra patterns, space-separated and possibly quoted
-i, --dist        Also clean the 'dist' dir
-t, --tox         Include '.tox' directory
-v, --venv        Include an existing virtualenv (in '.' or in '.venv')
-y, --bytecode    Also clean '.pyc', '.pyo', and package metadata
```

Note that `--all` is selective and only cleans out 'cheap' files; it especially excludes a local virtualenv that carries state (installed packages) you might not have recorded safely in requirements files yet, and the tree generated by `tox` that can take a while to reproduce.

The `--extra` options allows you to add any custom glob patterns to clean out.

#### Building the Project

Invoking `build` just delegates to `setup.py` right now. In the future, automatic detection of other project components like a Maven POM or Javascript build files might trigger additional build tools.

You can also include the `docs` task by adding the `--docs` option.

### 3.2.2 Documentation Tasks

#### Building Sphinx Documentation

*Rituals* provides automatic process management of a `sphinx-autobuild` daemon, which means you easily get a live-reload preview in your browser. To start the build watchdog, use `inv docs -w -b`. The `-b` means to open a new browser tab, after the server process is ready. To kill the server, call the `inv docs -k` command. You can check on the status of a running daemon with `inv docs -s`.

Note that sometimes you have to manually trigger a full rebuild via `inv docs --clean`, especially when you make structural changes (e.g. adding new chapters to the main toc-tree). Your browser will change the view to an empty canvas, just initiate a reload (`Ctrl-R`) when the build is done. Typically this is needed when the sidebar TOC is out of sync, which happens due to the optimizations in `sphinx-autobuild` that make it so responsive.

### 3.2.3 Release Workflow

#### Preparing a Release

`release-prep` performs QA checks, and switches to non-dev versioning.

**TODO**

#### Building a PEX Distribution

**TODO**

## 3.3 Configuration Reference

Please read Invoke's Configuration Guide on the concepts and basic mechanisms of its hierarchy of configuration files, environment variables, task namespaces and CLI flags. This reference guide lists the configuration options specific to tasks provided by *Rituals*.

---

**Note:** In the following tables of configurations settings, the root namespace of 'rituals' is implied, so to access them in a task you'd use `ctx.rituals.‹name›`, and `INVOKE_RITUALS_‹NAME›` to define an environment variable.

---

### 3.3.1 General Options

To make Python versions available that are not part of the host's default installation, `rituals.snakepits` is used, e.g. when performing multi-environment testing. The default is `/opt/pyenv/bin:/opt/pyrun/bin`.

| Name | Description |
|---|---|
| snakepits | Lookup path for Python interpreters |

### 3.3.2 Options for 'test'

If one of the directories in `rituals.snakepits` exists, it's added to the `PATH` of `tox`.

### 3.3.3 Options for 'docs'

The defaults for the `docs` task should almost always fit, but if you need to change them, you can.

| Name | Description |
|---|---|
| docs.sources | Documentation source folder (`docs`) |
| docs.build | Build area within the source folder (`_build`) |
| docs.watchdog.host | IP to bind `sphinx-autobuild` to (`127.0.0.1`) |
| docs.watchdog.port | Port to bind `sphinx-autobuild` to (`8840`) |

### 3.3.4 Options for 'release'

When `release.prep` changes the project configuration for a release and then tags the resulting changeset, the values from the following table are used for messages and names.

| Name | Description |
|---|---|
| release.commit.message | Message used (`:package: Release v{version}`) |
| release.tag.name | Release tag (`v{version}`) |
| release.tag.message | Tag annotation (`Release v{version}`) |

The `release.pex` task has an `--upload` option to upload the created archive to a WebDAV repository, e.g. a local Artifactory server or to Bintray. The best way to make this usable in each of your projects is to insert the base URL of your Python repository into your shell environment:

```
export INVOKE_RITUALS_RELEASE_UPLOAD_BASE_URL=\
"http://repo.example.com/artifactory/pypi-releases-local/"
```

| Name | Description |
|---|---|
| release.upload.base_url | WebDAV server end-point |
| release.upload.path | WebDAV path (`{name}/{version}/{filename}`) |

The following settings are used when building self-contained releases that integrate eGenix PyRun.

| Name | Description |
|---|---|
| pyrun.version | The version of *PyRun* to use (e.g. `2.1.0`) |
| pyrun.python | The version of *Python* to use (`2.6`, `2.7`, or `3.4`) |
| pyrun.ucs | Unicode code points size (`ucs2` or `ucs4`) |
| pyrun.platform | The platform ID (e.g. `linux-x86_64`, `macosx-10.5-x86_64`) |
| pyrun.base_url | Download location base URL pattern |
| pyrun.archive | Download location file name pattern |

The `rituals.pyrun.base_url` value can be a local `http[s]` URL of an Artifactory repository or some similar webserver, or else a `file://` URL of a file system cache. Note that you should keep the unchanged name of the original download location, i.e. do not change `rituals.pyrun.archive`. The location patterns can contain the `pyrun` settings as placeholders, e.g. `{version}`.

This sets a local download cache:

```
export INVOKE_RITUALS_PYRUN_BASE_URL="file://$HOME/Downloads"
```

You have to download the *PyRun* releases you plan to use to that directory, using your browser or `curl`.

### 3.3.5 Options for 'devpi'

When you call the `devpi.refresh` task without any option, the value of `rituals.devpi.requirements` is the name of the file parsed for the list of packages to refresh in the active `devpi` server. It defaults to `dev-requirements.txt`.

| Name | Description |
|---|---|
| devpi.requirements | Name of requirements file to use for refreshing |

## 3.4 Complete API Reference

The following is a complete API reference generated from source.

### 3.4.1 rituals package

Common tasks for 'Invoke' that are needed again and again.

The `rituals` package provides PyInvoke tasks that work for any project, based on its project metadata, to automate common developer chores like 'clean', 'build', 'dist', 'test', 'check', and 'release-prep' (for the moment).

The guiding principle for these tasks is to strictly separate low-level tasks for building and installing (via `setup.py`) from high-level convenience tasks a developer uses (via `tasks.py`). Invoke tasks can use Setuptools ones as building blocks, but never the other way 'round – this avoids bootstrapping head- aches during package installations using `pip`.

The easiest way to get a working project based on `rituals` is the `py-generic-project` cookiecutter template. That way you have a working project skeleton within minutes that is fully equipped, with all aspects of bootstrapping, building, testing, quality checks, continuous integration, documentation, and releasing covered. See here for more:

> https://github.com/Springerle/py-generic-project

Copyright 2015 Jürgen Hermann

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

The full LICENSE file and source are available at

> https://github.com/jhermann/rituals

## Subpackages

### rituals.acts package

rituals.acts – Task building blocks.

#### Submodules

**rituals.acts.basic module** Basic tasks.

**rituals.acts.devpi module** 'devpi' tasks.

`rituals.acts.devpi.`**`get_devpi_url`**(*ctx*)
Get currently used 'devpi' base URL.

**rituals.acts.documentation module** 'docs' tasks.

**class** `rituals.acts.documentation.`**`DocsUploader`**(*ctx*, *cfg*, *target*)
Bases: `object`

Helper to perform an upload of pre-built docs.

**`upload`**(*docs_base*, *release*)
Upload docs in `docs_base` to the target of this uploader.

`rituals.acts.documentation.`**`get_pypi_auth`**(*configfile=u'~/.pypirc'*)
Read auth from pip config.

rituals.acts.documentation.**watchdogctl**(*ctx*, *kill=False*, *verbose=True*)
Control / check a running Sphinx autobuild process.

**rituals.acts.github module** GitHub automation.

**rituals.acts.inspection module** 'docs' tasks.

**rituals.acts.pkgdeb module** 'deb' tasks.

**rituals.acts.releasing module** Release tasks.

rituals.acts.releasing.**get_egg_info**(*cfg*, *verbose=False*)
Call 'setup egg_info' and return the parsed meta-data.

**rituals.acts.testing module** Testing tasks.

### rituals.util package

rituals.util – Helper modules.

rituals.util.**add_dir2pypath**(*path*)
Add given directory to PYTHONPATH, e.g. for pylint.

rituals.util.**search_file_upwards**(*name*, *base=None*)
Search for a file named *name* from cwd or given directory to root. Return None if nothing's found.

#### Subpackages

**rituals.util.scm package** rituals.util.scm – Source Code Management support.

rituals.util.scm.**auto_detect**(*workdir*)
Return string signifying the SCM used in the given directory.

Currently, 'git' is supported. Anything else returns 'unknown'.

rituals.util.scm.**provider**(*workdir*, *commit=True*, ***kwargs*)
Factory for the correct SCM provider in *workdir*.

#### Submodules

**rituals.util.scm.base module** Provider base class.

**class** rituals.util.scm.base.**ProviderBase**(*workdir*, *commit=True*, ***kwargs*)
Bases: object

Base class for SCM providers.

**run**(*cmd*, **args*, ***kwargs*)
Run a command.

**run_elective**(*cmd*, **args*, ***kwargs*)
Run a command, or just echo it, depending on *commit*.

**rituals.util.scm.git module** git SCM provider.

**class** rituals.util.scm.git.**GitProvider**(*workdir*, *commit=True*, ***kwargs*)
Bases: *rituals.util.scm.base.ProviderBase*

git SCM provider.

Expects a working *git* executable in the path, having a reasonably current version.

**add_file**(*filename*)
Stage a file for committing.

**commit**(*message*)
Commit pending changes.

**key = u'git'**

**pep440_dev_version**(*verbose=False*, *non_local=False*)
Return a PEP-440 dev version appendix to the main version number.

Result is None if the workdir is in a release-ready state (i.e. clean and properly tagged).

**tag**(*label*, *message=None*)
Tag the current workdir state.

**workdir_is_clean**(*quiet=False*)
Check for uncommitted changes, return *True* if everything is clean.

Inspired by http://stackoverflow.com/questions/3878624/.

**rituals.util.scm.null module** Provider for unknown SCM systems.

**class** rituals.util.scm.null.**NullProvider**(*workdir*, *commit=True*, ***kwargs*)
Bases: *rituals.util.scm.base.ProviderBase*

Stub provider for unknown SCM systems.

This implements the provider interface, mostly emitting warnings.

**add_file**(*filename*)
Stage a file for committing, or commit it directly (depending on the SCM).

**commit**(*message*)
Commit pending changes.

**key = u'unknown'**

**pep440_dev_version**(*verbose=False*, *non_local=False*)
Return a PEP-440 dev version appendix to the main version number.

**tag**(*label*, *message=None*)
Tag the current workdir state.

**workdir_is_clean**(*quiet=False*)
Check for uncommitted changes, return *True* if everything is clean.

**Submodules**

**rituals.util.antglob module** Recursive globbing with ant-style syntax.

**class** rituals.util.antglob.**FileSet**(*root*, *patterns*)
Bases: object

Ant-style file and directory matching.

Produces an iterator of all of the files that match the provided patterns. Note that directory matches must end with a slash, and if they're exclusions, they won't be scanned (which prunes anything in that directory that would otherwise match).

**Directory specifiers:** ** matches zero or more directories. / path separator.

**File specifiers:**

- glob style wildcard.

[chars] inclusive character sets. [^chars] exclusive character sets.

#### Examples

**/*.py recursively match all python files. foo/**/*.py recursively match all python files in the 'foo' directory. *.py match all the python files in the current directory. */*.txt match all the text files in top-level directories. foo/**/* all files under directory 'foo'. /*/ top-level directories. foo/ the directory 'foo' itself. */foo/ any directory named 'foo'. **/.* hidden files. **/.*/ hidden directories.

**included**(*path*, *is_dir=False*)
Check patterns in order, last match that includes or excludes *path* wins. Return *None* on undecided.

**walk**(***kwargs*)
Like *os.walk* and taking the same keyword arguments, but generating paths relative to the root.

Starts in the fileset's root and filters based on its patterns. If `with_root=True` is passed in, the generated paths include the root path.

rituals.util.antglob.**includes**(*pattern*)
A single inclusive glob pattern.

rituals.util.antglob.**excludes**(*pattern*)
A single exclusive glob pattern.

**rituals.util.filesys module** File system helpers.

rituals.util.filesys.**pretty_path**(*path*, *_home_re=<_sre.SRE_Pattern object>*)
Prettify path for humans, and make it Unicode.

rituals.util.filesys.**pushd**(**args*, ***kwds*)
A context that enters a given directory and restores the old state on exit.

The original directory is returned as the context variable.

rituals.util.filesys.**url_as_file**(**args*, ***kwds*)
Context manager that GETs a given *url* and provides it as a local file.

The file is in a closed state upon entering the context, and removed when leaving it, if still there.

To give the file name a specific extension, use *ext*; the extension can optionally include a separating dot, otherwise it will be added.

**Parameters**

- **url** (*str*) – URL to retrieve.

- **ext** (*str, optional*) – Extension for the generated filename.

**Yields** *str* – The path to a temporary file with the content of the URL.

**Raises** `requests.RequestException` – Base exception of `requests`, see its docs for more detailed ones.

**Example**

```
>>> import io, re, json
>>> with url_as_file('https://api.github.com/meta', ext='json') as meta:
...     meta, json.load(io.open(meta, encoding='ascii'))['hooks']
(u'/tmp/www-api.github.com-Ba5OhD.json', [u'192.30.252.0/22'])
```

**rituals.util.notify module** Log notification messages to console.

`rituals.util.notify.`**`banner`**(*msg*)
Emit a banner just like Invoke's *run(..., echo=True)*.

`rituals.util.notify.`**`error`**(*msg*)
Emit an error message to stderr.

`rituals.util.notify.`**`failure`**(*msg*)
Emit a fatal message and exit.

`rituals.util.notify.`**`info`**(*msg*)
Emit a normal message.

`rituals.util.notify.`**`warning`**(*msg*)
Emit a warning message.

**rituals.util.shell module** Shell command calls.

`rituals.util.shell.`**`capture`**(*cmd*, ***kw*)
Run a command and return its stripped captured output.

`rituals.util.shell.`**`run`**(*cmd*, ***kw*)
Run a command and flush its output.

**rituals.util.which module** Find the full path to commands.

**which(command, path=None, verbose=0, exts=None)** Return the full path to the first match of the given command on the path.

**whichall(command, path=None, verbose=0, exts=None)** Return a list of full paths to all matches of the given command on the path.

**whichgen(command, path=None, verbose=0, exts=None)** Return a generator which will yield full paths to all matches of the given command on the path.

By default the PATH environment variable is searched (as well as, on Windows, the AppPaths key in the registry), but a specific 'path' list to search may be specified as well. On Windows, the PATHEXT environment variable is applied as appropriate.

**If "verbose" is true then a tuple of the form** (<fullpath>, <matched-where-description>)

is returned for each match. The latter element is a textual description of where the match was found. For example:

from PATH element 0 from HKLMSOFTWARE...perl.exe

**exception** rituals.util.which.**WhichError**
Bases: exceptions.Exception

Executable not found by *which* module.

rituals.util.which.**which**(*command*, *path=None*, *verbose=0*, *exts=None*)
Return the full path to the first match of the given command on the path.

"command" is a the name of the executable to search for. "path" is an optional alternate path list to search. The default it

to use the PATH environment variable.

**"verbose", if true, will cause a 2-tuple to be returned. The second** element is a textual description of where the match was found.

**"exts" optionally allows one to specify a list of extensions to use** instead of the standard list for this system. This can effectively be used as an optimization to, for example, avoid stat's of "foo.vbs" when searching for "foo" and you know it is not a VisualBasic script but ".vbs" is on PATHEXT. This option is only supported on Windows.

If no match is found for the command, a WhichError is raised.

rituals.util.which.**whichall**(*command*, *path=None*, *verbose=0*, *exts=None*)
Return a list of full paths to all matches of the given command on the path.

"command" is a the name of the executable to search for. "path" is an optional alternate path list to search. The default it

to use the PATH environment variable.

**"verbose", if true, will cause a 2-tuple to be returned for each** match. The second element is a textual description of where the match was found.

**"exts" optionally allows one to specify a list of extensions to use** instead of the standard list for this system. This can effectively be used as an optimization to, for example, avoid stat's of "foo.vbs" when searching for "foo" and you know it is not a VisualBasic script but ".vbs" is on PATHEXT. This option is only supported on Windows.

rituals.util.which.**whichgen**(*command*, *path=None*, *verbose=0*, *exts=None*)
Return a generator of full paths to the given command.

"command" is a the name of the executable to search for. "path" is an optional alternate path list to search. The default it

to use the PATH environment variable.

**"verbose", if true, will cause a 2-tuple to be returned for each** match. The second element is a textual description of where the match was found.

**"exts" optionally allows one to specify a list of extensions to use** instead of the standard list for this system. This can effectively be used as an optimization to, for example, avoid stat's of "foo.vbs" when searching for "foo" and you know it is not a VisualBasic script but ".vbs" is on PATHEXT. This option is only supported on Windows.

This method returns a generator which yields either full paths to the given command or, if verbose, tuples of the form (<path to command>, <where path found>).

### Submodules

#### rituals.config module

Project configuration and layout.

rituals.config.**get_project_root**()
Determine location of *tasks.py*.

rituals.config.**load**()
Load and return configuration as a Bunch.

Values are based on DEFAULTS, and metadata from setup.py.

rituals.config.**set_flat_layout**()
Switch default project layout to everything top-level.

rituals.config.**set_maven_layout**()
Switch default project layout to Maven-like.

#### rituals.easy module

Default namespace for convenient wildcard import in task definition modules.

**class** rituals.easy.**Collection**(**args*, ***kwargs*)
Bases: object

A collection of executable tasks.

**__getitem__**(*name=None*)
Returns task named name. Honors aliases and subcollections.

If this collection has a default task, it is returned when name is empty or None. If empty input is given and no task has been selected as the default, ValueError will be raised.

Tasks within subcollections should be given in dotted form, e.g. 'foo.bar'. Subcollection default tasks will be returned on the subcollection's name.

**add_collection**(*coll*, *name=None*)
Add .*Collection* coll as a sub-collection of this one.

**Parameters**

- **coll** – The .*Collection* to add.
- **name** (*str*) – The name to attach the collection as. Defaults to the collection's own internal name.

**add_task**(*task*, *name=None*, *default=None*)
Add .*Task* task to this collection.

**Parameters**

- **task** – The .*Task* object to add to this collection.
- **name** – Optional string name to bind to (overrides the task's own self-defined name attribute and/or any Python identifier (i.e. .func_name.)
- **default** – Whether this task should be the collection default.

**configuration**(*taskpath=None*)
Obtain merged configuration values from collection & children.

**Note:** Merging uses `copy.deepcopy` to prevent state bleed.

**Parameters** **taskpath** – (Optional) Task name/path, identical to that used for *~.Collection.__getitem__* (e.g. may be dotted for nested tasks, etc.) Used to decide which path to follow in the collection tree when merging config values.

**Returns** A *dict* containing configuration values.

**configure**(*options*)

(Recursively) merge `options` into the current *.configuration*.

Options configured this way will be available to all `contextualized tasks`. It is recommended to use unique keys to avoid potential clashes with other config options

For example, if you were configuring a Sphinx docs build target directory, it's better to use a key like `'sphinx.target'` than simply `'target'`.

**Parameters** **options** – An object implementing the dictionary protocol.

**Returns** `None`.

**classmethod from_module**(*module*, *name=None*, *config=None*, *loaded_from=None*)

Return a new *.Collection* created from `module`.

Inspects `module` for any *.Task* instances and adds them to a new *.Collection*, returning it. If any explicit namespace collections exist (named `ns` or `namespace`) a copy of that collection object is preferentially loaded instead.

When the implicit/default collection is generated, it will be named after the module's `__name__` attribute, or its last dotted section if it's a submodule. (I.e. it should usually map to the actual `.py` filename.)

Explicitly given collections will only be given that module-derived name if they don't already have a valid `.name` attribute.

**Parameters**

- **name** (*str*) – A string, which if given will override any automatically derived collection name (or name set on the module's root namespace, if it has one.)
- **config** (*dict*) – Used to set config options on the newly created *.Collection* before returning it (saving you a call to *.configure*.)

  If the imported module had a root namespace object, `config` is merged on top of it (i.e. overriding any conflicts.)
- **loaded_from** (*str*) – Identical to the same-named kwarg from the regular class constructor - should be the path where the module was found.

**split_path**(*path*)

Obtain first collection + remainder, of a task path.

E.g. for `"subcollection.taskname"`, return `("subcollection", "taskname")`; for `"subcollection.nested.taskname"` return `("subcollection", "nested.taskname")`, etc.

An empty path becomes simply `('', '')`.

**subtask_name**(*collection_name*, *task_name*)

**task_names**

Return all task identifiers for this collection as a dict.

Specifically, a dict with the primary/"real" task names as the key, and any aliases as a list value.

**task_with_config**(*name*)
Return task named `name` plus its configuration dict.

E.g. in a deeply nested tree, this method returns the *.Task*, and a configuration dict created by merging that of this *.Collection* and any nested *Collections* <*.Collection*>, up through the one actually holding the *.Task*.

See ~*.Collection.__getitem__* for semantics of the `name` argument.

**Returns** Two-tuple of (*.Task*, *dict*).

**to_contexts**()
Returns all contained tasks and subtasks as a list of parser contexts.

`rituals.easy.`**task**(**args*, ***kwargs*)
Wrapper for *.task* which sets `contextualized=True` by default.

Please see *.task* for documentation.

`rituals.easy.`**pushd**(**args*, ***kwds*)
A context that enters a given directory and restores the old state on exit.

The original directory is returned as the context variable.

## 3.5 Contribution Guidelines

### 3.5.1 Overview

Contributing to this project is easy, and reporting an issue or adding to the documentation also improves things for every user. You don't need to be a developer to contribute.

#### Reporting issues

Please use the *GitHub issue tracker*, and describe your problem so that it can be easily reproduced. Providing relevant version information on the project itself and your environment helps with that.

#### Improving documentation

The easiest way to provide examples or related documentation that helps other users is the *GitHub wiki*.

If you are comfortable with the Sphinx documentation tool, you can also prepare a pull request with changes to the core documentation. GitHub's built-in text editor makes this especially easy, when you choose the *"Create a new branch for this commit and start a pull request"* option on saving. Small fixes for typos and the like are a matter of minutes when using that tool.

#### Code contributions

Here's a quick guide to improve the code:

1. Fork the repo, and clone the fork to your machine.
2. Add your improvements, the technical details are further below.
3. Run the tests and make sure they're passing (`invoke test`).
4. Check for violations of code conventions (`invoke check`).

5. Make sure the documentation builds without errors (`invoke build --docs`).
6. Push to your fork and submit a pull request.

Please be patient while waiting for a review. Life & work tend to interfere.

### 3.5.2 Details on contributing code

This project is written in Python, and the documentation is generated using Sphinx. setuptools and Invoke are used to build and manage the project. Tests are written and executed using pytest and tox.

#### Set up a working development environment

To set up a working directory from your own fork, follow these steps, but replace the repository `https` URLs with SSH ones that point to your fork.

For that to work on Debian type systems, you need the `git`, `python`, and `python-virtualenv` packages installed. Other distributions are similar.

#### Add your changes to a feature branch

For any cohesive set of changes, create a *new* branch based on the current upstream `master`, with a name reflecting the essence of your improvement.

```
git branch "name-for-my-fixes" origin/master
git checkout "name-for-my-fixes"
... make changes...
invoke ci # check output for broken tests, or PEP8 violations and the like
... commit changes...
git push origin "name-for-my-fixes"
```

Please don't create large lumps of unrelated changes in a single pull request. Also take extra care to avoid spurious changes, like mass whitespace diffs. All Python sources use spaces to indent, not TABs.

#### Make sure your changes work

Some things that will increase the chance that your pull request is accepted:

- Follow style conventions you see used in the source already (and read PEP8).
- Include tests that fail *without* your code, and pass *with* it. Only minor refactoring and documentation changes require no new tests. If you are adding functionality or fixing a bug, please also add a test for it!
- Update any documentation or examples impacted by your change.
- Styling conventions and code quality are checked with `invoke check`, tests are run using `invoke test`, and the docs can be built locally using `invoke build --docs`.

Following these hints also expedites the whole procedure, since it avoids unnecessary feedback cycles.

## 3.6 Software License

Copyright 2015 Jürgen Hermann

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

The full LICENSE file and source are available at

https://github.com/jhermann/rituals

### 3.6.1 Full License Text

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc., <http://fsf.org/>
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.
```

```
  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
```

```
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
```

```
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
```

```
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
```

```
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    {description}
    Copyright (C) {year}  {fullname}

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License version 2 as
    published by the Free Software Foundation.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  {signature of Ty Coon}, 1 April 1989
  Ty Coon, President of Vice
```

```
This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

CHAPTER 4

# References

## 4.1 Tools

- Cookiecutter
- PyInvoke
- pytest
- tox
- Pylint
- twine
- bpython
- yolk3k

CHAPTER 5

# Indices and Tables

- genindex
- modindex
- search

## r

rituals, 10
rituals.acts, 11
rituals.acts.basic, 11
rituals.acts.devpi, 11
rituals.acts.documentation, 11
rituals.acts.github, 12
rituals.acts.inspection, 12
rituals.acts.pkgdeb, 12
rituals.acts.releasing, 12
rituals.acts.testing, 12
rituals.config, 17
rituals.easy, 17
rituals.util, 12
rituals.util.antglob, 14
rituals.util.filesys, 14
rituals.util.notify, 15
rituals.util.scm, 12
rituals.util.scm.base, 12
rituals.util.scm.git, 13
rituals.util.scm.null, 13
rituals.util.shell, 15
rituals.util.which, 15

## Symbols

__getitem__() (rituals.easy.Collection method), 17

## A

add_collection() (rituals.easy.Collection method), 17
add_dir2pypath() (in module rituals.util), 12
add_file() (rituals.util.scm.git.GitProvider method), 13
add_file() (rituals.util.scm.null.NullProvider method), 13
add_task() (rituals.easy.Collection method), 17
auto_detect() (in module rituals.util.scm), 12

## B

banner() (in module rituals.util.notify), 15

## C

capture() (in module rituals.util.shell), 15
Collection (class in rituals.easy), 17
commit() (rituals.util.scm.git.GitProvider method), 13
commit() (rituals.util.scm.null.NullProvider method), 13
configuration() (rituals.easy.Collection method), 17
configure() (rituals.easy.Collection method), 18

## D

DocsUploader (class in rituals.acts.documentation), 11

## E

error() (in module rituals.util.notify), 15
excludes() (in module rituals.util.antglob), 14

## F

failure() (in module rituals.util.notify), 15
FileSet (class in rituals.util.antglob), 14
from_module() (rituals.easy.Collection class method), 18

## G

get_devpi_url() (in module rituals.acts.devpi), 11
get_egg_info() (in module rituals.acts.releasing), 12
get_project_root() (in module rituals.config), 17
get_pypi_auth() (in module rituals.acts.documentation), 11
GitProvider (class in rituals.util.scm.git), 13

## I

included() (rituals.util.antglob.FileSet method), 14
includes() (in module rituals.util.antglob), 14
info() (in module rituals.util.notify), 15

## K

key (rituals.util.scm.git.GitProvider attribute), 13
key (rituals.util.scm.null.NullProvider attribute), 13

## L

load() (in module rituals.config), 17

## N

NullProvider (class in rituals.util.scm.null), 13

## P

pep440_dev_version() (rituals.util.scm.git.GitProvider method), 13
pep440_dev_version() (rituals.util.scm.null.NullProvider method), 13
pretty_path() (in module rituals.util.filesys), 14
provider() (in module rituals.util.scm), 12
ProviderBase (class in rituals.util.scm.base), 12
pushd() (in module rituals.easy), 19
pushd() (in module rituals.util.filesys), 14

## R

rituals (module), 10
rituals.acts (module), 11
rituals.acts.basic (module), 11
rituals.acts.devpi (module), 11
rituals.acts.documentation (module), 11
rituals.acts.github (module), 12
rituals.acts.inspection (module), 12
rituals.acts.pkgdeb (module), 12
rituals.acts.releasing (module), 12
rituals.acts.testing (module), 12
rituals.config (module), 17

rituals.easy (module), 17
rituals.util (module), 12
rituals.util.antglob (module), 14
rituals.util.filesys (module), 14
rituals.util.notify (module), 15
rituals.util.scm (module), 12
rituals.util.scm.base (module), 12
rituals.util.scm.git (module), 13
rituals.util.scm.null (module), 13
rituals.util.shell (module), 15
rituals.util.which (module), 15
run() (in module rituals.util.shell), 15
run() (rituals.util.scm.base.ProviderBase method), 12
run_elective() (rituals.util.scm.base.ProviderBase method), 12

## S

search_file_upwards() (in module rituals.util), 12
set_flat_layout() (in module rituals.config), 17
set_maven_layout() (in module rituals.config), 17
split_path() (rituals.easy.Collection method), 18
subtask_name() (rituals.easy.Collection method), 18

## T

tag() (rituals.util.scm.git.GitProvider method), 13
tag() (rituals.util.scm.null.NullProvider method), 13
task() (in module rituals.easy), 19
task_names (rituals.easy.Collection attribute), 18
task_with_config() (rituals.easy.Collection method), 18
to_contexts() (rituals.easy.Collection method), 19

## U

upload() (rituals.acts.documentation.DocsUploader method), 11
url_as_file() (in module rituals.util.filesys), 14

## W

walk() (rituals.util.antglob.FileSet method), 14
warning() (in module rituals.util.notify), 15
watchdogctl() (in module rituals.acts.documentation), 11
which() (in module rituals.util.which), 16
whichall() (in module rituals.util.which), 16
WhichError, 15
whichgen() (in module rituals.util.which), 16
workdir_is_clean() (rituals.util.scm.git.GitProvider method), 13
workdir_is_clean() (rituals.util.scm.null.NullProvider method), 13